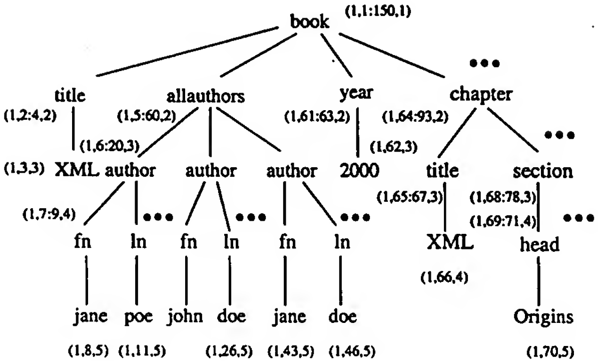

book
(1,1:150,1)
•••
title
(1,2:4,2)
allauthors
(1,5:60,2)
year
(1,61:63,2)
chapter
(1,64:93,2)
•••
(1,6:20,3)
(1,62,3)
(1,3,3)
XML
author
author
author
2000
title
section
(1,65:67,3)
(1,68:78,3)
(1,69:71,4)
(1,7:9,4)
•••
•••
•••
•••
fn
ln
fn
ln
fn
ln
XML
head
(1,66,4)
jane
poe
john
doe
jane
doe
Origins
(1,8,5)
(1,11,5)
(1,26,5)
(1,43,5)
(1,46,5)
(1,70,5)

FIG. 1

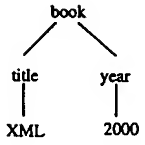


FIG. 2A

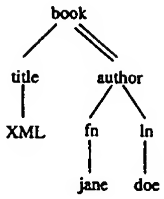


FIG. 2B

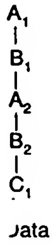


FIG. 3A

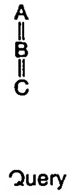


FIG. 3B

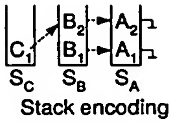


FIG. 3C

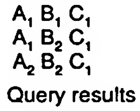


FIG. 3D

```
Algorithm PathStack($q$)
while $\neg$end($q$)
  $q_{min}$ = getMinSource($q$)
  for $q_i$ in subtreeNodes($q$) // clean stacks
    while ($\neg$empty($S_{q_i}$) $\wedge$ topR($S_{q_i}$) $<$ nextL($T_{q_{min}}$))
      pop($S_{q_i}$)
  moveStreamToStack($T_{q_{min}}$, $S_{q_{min}}$, pointer to
                                        top($S_{parent(q_{min})}$))
  if (isLeaf($q_{min}$))
    showSolutions($S_{q_{min}}$, 1)
    pop($S_{q_{min}}$)

Function end($q$)
  return $\forall q_i \in$ subtreeNodes($q$) : isLeaf($q_i$) $\Rightarrow$ eof($T_{q_i}$)

Function getMinSource($q$)
  return $q_i \in$ subtreeNodes($q$) such that nextL($T_{q_i}$)
    is minimal

Procedure moveStreamToStack($T_q$, $S_q$, $p$)
push($S_q$, (next($T_q$), $p$))
advance($T_q$)
```

PathStack

FIG. 4

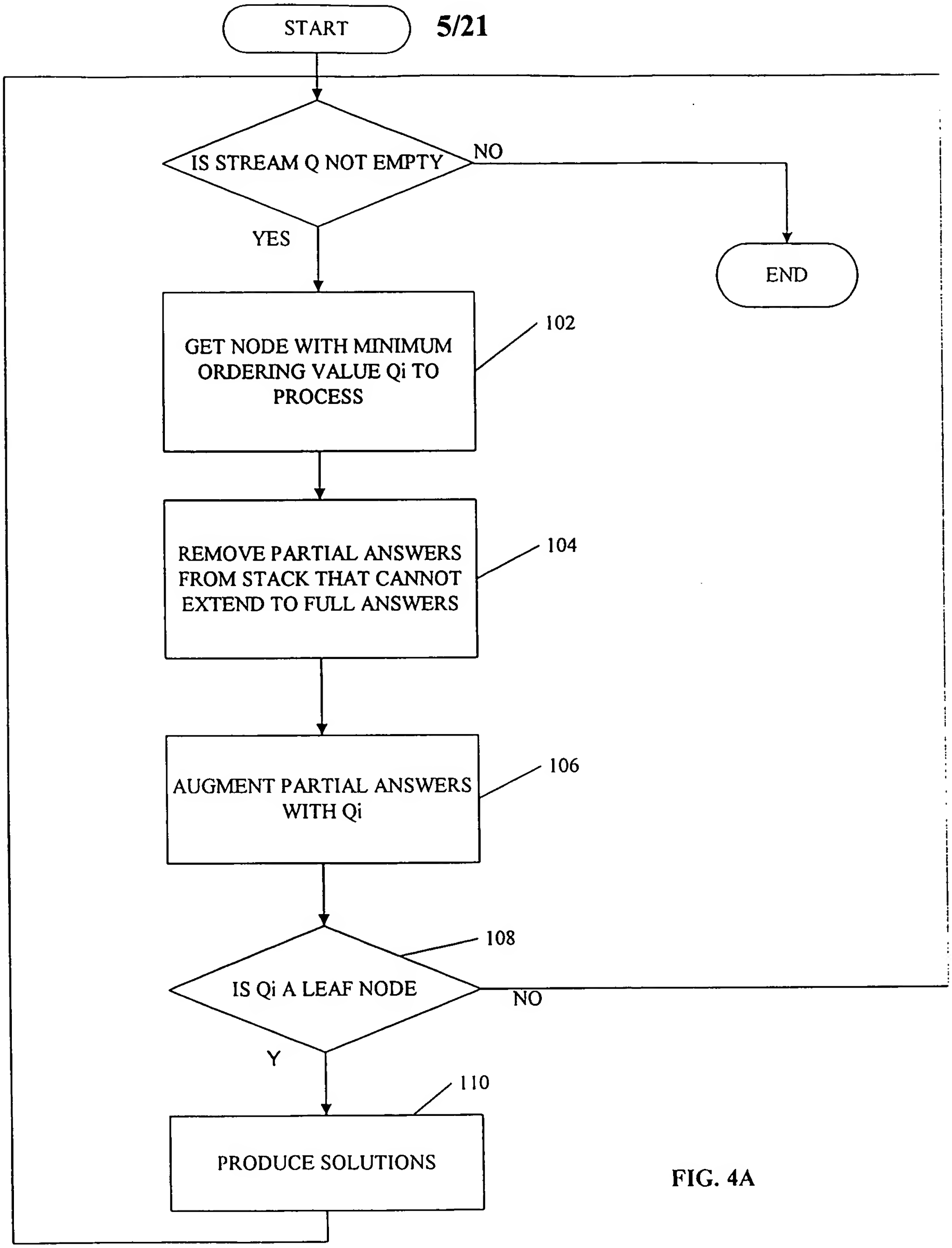

START
IS STREAM Q NOT EMPTY
NO
YES
END
GET NODE WITH MINIMUM ORDERING VALUE Qi TO PROCESS
102
REMOVE PARTIAL ANSWERS FROM STACK THAT CANNOT EXTEND TO FULL ANSWERS
104
AUGMENT PARTIAL ANSWERS WITH Qi
106
IS Qi A LEAF NODE
108
NO
Y
110
PRODUCE SOLUTIONS

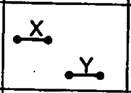
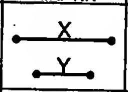
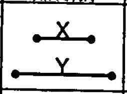
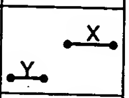
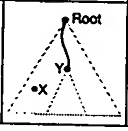
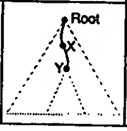
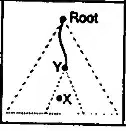
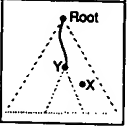
FIG. 4A

```
Procedure showSolutions($SN$,$SP$)
// Assume, for simplicity, that the stacks of the query
//   nodes from the root to the current leaf node we
//   are interested in can be accessed as $S[1],\ldots,S[n]$.
// Also assume that we have a global array index[1..$n$]
//   of pointers to the stack elements.
// index[$i$] represents the position in the $i$'th stack that
//   we are interested in for the current solution, where
//   the bottom of each stack has position 1.

// Mark we are interested in position $SP$ of stack $SN$.
index[$SN$] = $SP$
if ($SN == 1$) // we are in the root
  // output solutions from the stacks
  output ($S[n]$.index[$n$],...,$S[1]$.index[1])
else // recursive call
  for $i = 1$ to $S[SN]$.index[$SN$].pointer_to_parent
    showSolutions($SN - 1, i$)
```

Procedure showSolutions

FIG 5

| | Case 1 | Case 2 | Case 3 | Case 4 |
|---|---|---|---|---|
| Property | X.R<Y.L | X.L<Y.L<br>X.R>Y.R | X.L>Y.L<br>X.R<Y.R | X.L>Y.R |
| Segments |  |  |  |  |
| Tree |  |  |  |  |

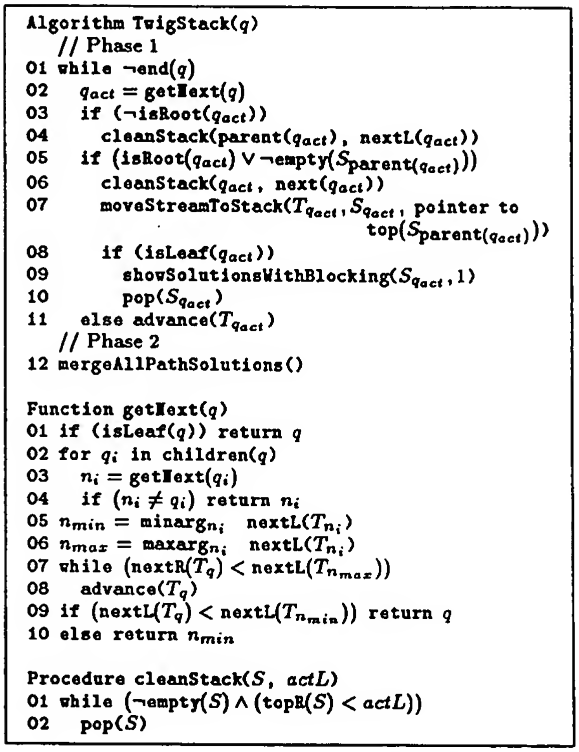
Cases for PathStack and TwigStack

FIG 6

```
Algorithm PathMPMJ($q$)
while ($\neg$eof($T_q$)$\wedge$ (isRoot($q$)$\vee$
                    nextL($q$) < nextR(parent($q$))))
   for ($q_i \in$ subtreeNodes($q$)) // advance descendants
      while (nextL($q_i$) < nextL(parent($q_i$)))
         advance($T_{q_i}$)
      PushMark($T_{q_i}$)
   if (isLeaf($q$)) // solution in the streams' heads
      outputSolution()
   else PathMPMJ(child($q$))
   advance($T_q$)
   for ($q_i \in$ subtreeNodes($q$)) // backtrack descendants
      PopMark($T_{q_i}$)
```

PathMPMJ

FIG. 7



```
Algorithm TwigStack($q$)
  // Phase 1
while $\neg$end($q$)
  $q_{act}$ = getNext($q$)
  if ($\neg$isRoot($q_{act}$))
    cleanStack(parent($q_{act}$), nextL($q_{act}$))
  if (isRoot($q_{act}$) $\vee$ $\neg$empty($S_{\text{parent}(q_{act})}$))
    cleanStack($q_{act}$, next($q_{act}$))
    moveStreamToStack($T_{q_{act}}$, $S_{q_{act}}$, pointer to
                                        top($S_{\text{parent}(q_{act})}$))
    if (isLeaf($q_{act}$))
      showSolutionsWithBlocking($S_{q_{act}}$, 1)
      pop($S_{q_{act}}$)
  else advance($T_{q_{act}}$)
  // Phase 2
mergeAllPathSolutions()

Function getNext($q$)
if (isLeaf($q$)) return $q$
for $q_i$ in children($q$)
  $n_i$ = getNext($q_i$)
  if ($n_i \neq q_i$) return $n_i$
$n_{min}$ = minarg$_{n_i}$ nextL($T_{n_i}$)
$n_{max}$ = maxarg$_{n_i}$ nextL($T_{n_i}$)
while (nextR($T_q$) < nextL($T_{n_{max}}$))
  advance($T_q$)
if (nextL($T_q$) < nextL($T_{n_{min}}$)) return $q$
else return $n_{min}$

Procedure cleanStack($S$, $actL$)
while ($\neg$empty($S$) $\wedge$ (topR($S$) < $actL$))
  pop($S$)
```

TwigStack

FIG. 8

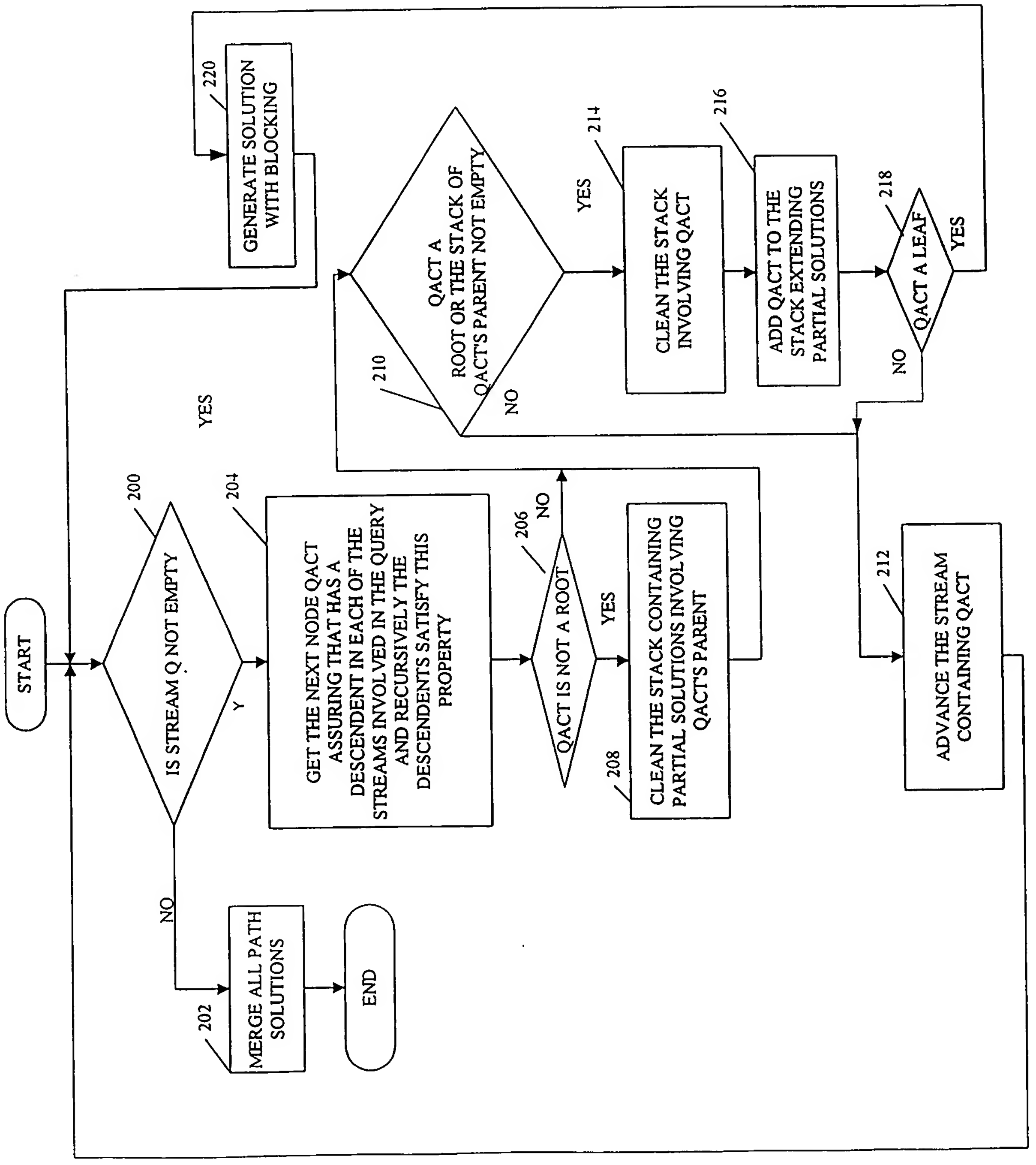

START
IS STREAM Q NOT EMPTY
200
YES
Y
NO
MERGE ALL PATH SOLUTIONS
202
END
GET THE NEXT NODE QACT ASSURING THAT HAS A DESCENDENT IN EACH OF THE STREAMS INVOLVED IN THE QUERY AND RECURSIVELY THE DESCENDENTS SATISFY THIS PROPERTY
204
QACT IS NOT A ROOT
206
NO
YES
CLEAN THE STACK CONTAINING PARTIAL SOLUTIONS INVOLVING QACT'S PARENT
208
GENERATE SOLUTION WITH BLOCKING
220
QACT A ROOT OR THE STACK OF QACT'S PARENT NOT EMPTY
210
NO
YES
CLEAN THE STACK INVOLVING QACT
214
ADD QACT TO THE STACK EXTENDING PARTIAL SOLUTIONS
216
QACT A LEAF
218
YES
NO
ADVANCE THE STREAM CONTAINING QACT
212

FIG. 8A

```
Algorithm TwigStackXB($q$)
01 while $\neg$end($q$)
02   $q_{act}$ = getNext($q$)
(03) if (isPlainValue($T_{q_{act}}$))
04     if ($\neg$isRoot($q_{act}$))
05       cleanStack(parent($q_{act}$), next($q_{act}$))
06     if (isRoot($q_{act}$) $\vee$ $\neg$empty($S_{\text{parent}(q_{act})}$))
07       cleanStack($q_{act}$, next($q_{act}$))
08       moveStreamToStack($T_{q_{act}}$, $S_{q_{act}}$, pointer to
                                   top($S_{\text{parent}(q_{act})}$))
09       if (isLeaf($q_{act}$))
10         showSolutionsWithBlocking($S_{q_{act}}$, 1)
11         pop($S_{q_{act}}$)
12     else advance($T_{q_{act}}$)
(13) else if ($\neg$isRoot($q_{act}$) $\wedge$ empty($S_{\text{parent}(q_{act})}$)$\wedge$
              nextL($T_{\text{parent}(q_{act})}$) > nextR($T_{q_{act}}$))
(14)   advance($T_{q_{act}}$) // Not part of a solution
(15) else // Might have a child in some solution
(16)   drillDown($T_{q_{act}}$)
   // Phase 2
17 mergeAllPathSolutions()

Function getNext($q$)
01 if (isLeaf($q$)) return $q$
02 for $q_i$ in children($q$)
03   $n_i$ = getNext($q_i$)
(04) if ($q_i \neq n_i$ $\vee$ $\neg$isPlainValue($T_{n_i}$)) return $n_i$
05 $n_{min}$ = minarg$_{n_i}$ nextL($T_{n_i}$)
06 $n_{max}$ = maxarg$_{n_i}$ nextL($T_{n_i}$)
07 while (nextR($T_q$) < nextL($T_{n_{max}}$))
08   advance($T_q$)
09 if (nextL($T_q$) < nextL($T_{n_{min}}$)) return $q$
10 else return $n_{min}$

Procedure cleanStack($S$, $actL$)
01 while ($\neg$empty($S$) $\wedge$ (topR($S$) < $actL$))
02   pop($S$)
```

TwigStackXB

FIG. 9

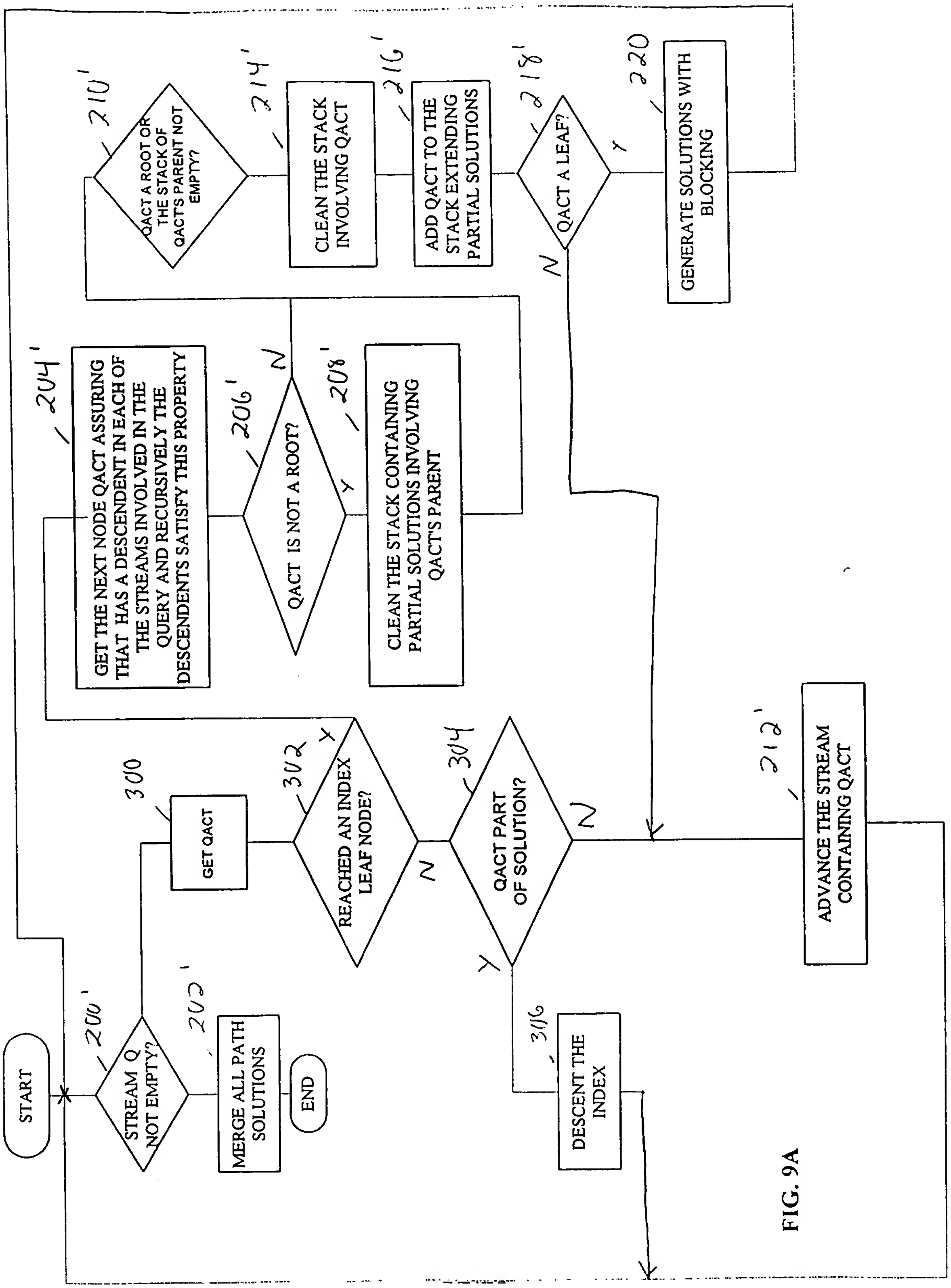
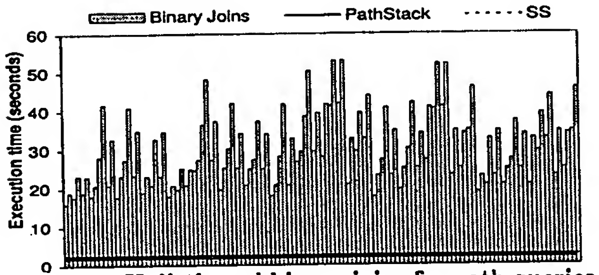
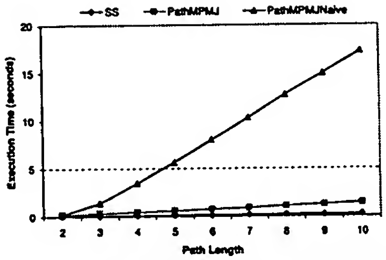

START
STREAM Q NOT EMPTY?
200'
MERGE ALL PATH SOLUTIONS
202'
END
GET QACT
300
REACHED AN INDEX LEAF NODE?
302
Y
N
QACT PART OF SOLUTION?
304
Y
N
DESCENT THE INDEX
306
ADVANCE THE STREAM CONTAINING QACT
212'
GET THE NEXT NODE QACT ASSURING THAT HAS A DESCENDENT IN EACH OF THE STREAMS INVOLVED IN THE QUERY AND RECURSIVELY THE DESCENDENTS SATISFY THIS PROPERTY
204'
QACT IS NOT A ROOT?
206'
Y
N
CLEAN THE STACK CONTAINING PARTIAL SOLUTIONS INVOLVING QACT'S PARENT
208'
QACT A ROOT OR THE STACK OF QACT'S PARENT NOT EMPTY?
210'
CLEAN THE STACK INVOLVING QACT
214'
ADD QACT TO THE STACK EXTENDING PARTIAL SOLUTIONS
216'
QACT A LEAF?
218'
Y
N
GENERATE SOLUTIONS WITH BLOCKING
220

FIG. 9A



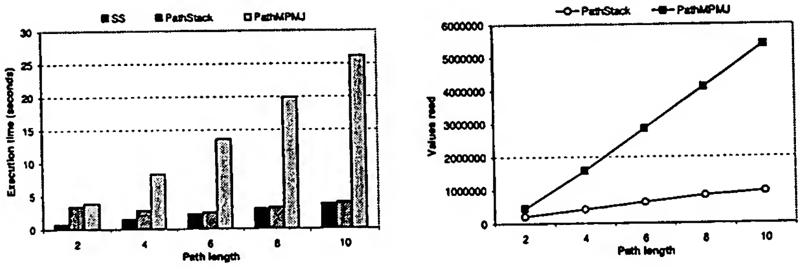
Holistic and binary joins for path queries

FIG. 10



PathMPMJ versus PathMPMJNaive

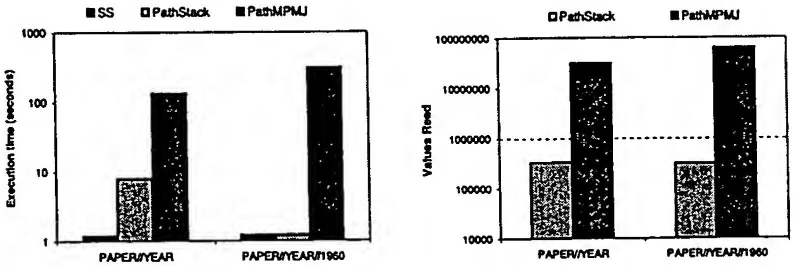
FIG. 11



(a) Execution time (b) Number of elements read

PathStack versus PathMPMJ using synthetic data sets

FIG. 12A FIG. 12B



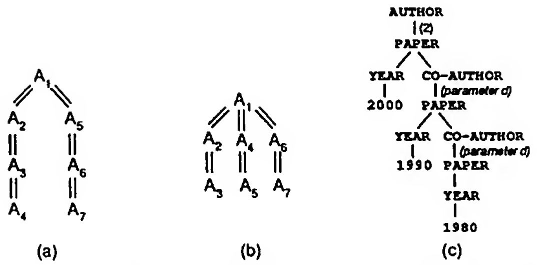
(a) Execution time

(b) Number of elements read

PathStack versus PathMPMJ for the unfolded DBLP data set

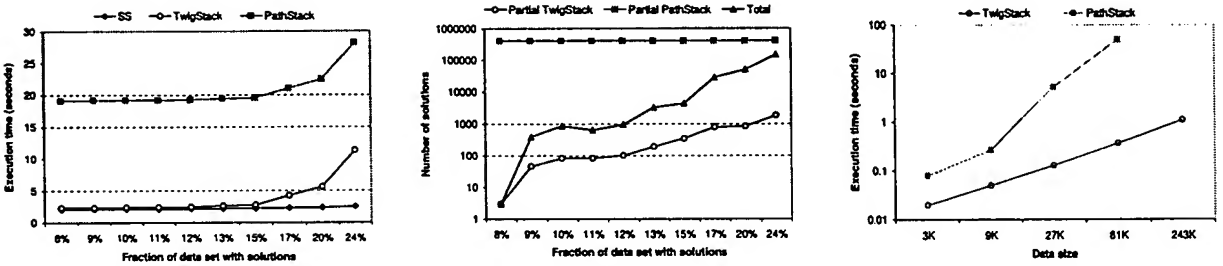
FIG. 13A

FIG 13B



Twig queries used in the experiments

FIG. 14A FIG. 14B FIG. 14C



(a) Execution time

(b) Number of solutions

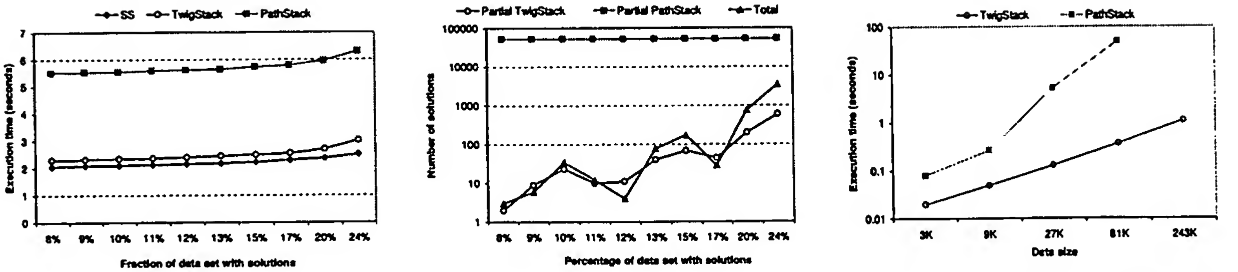
(c) Execution time for a complex query

PathStack versus TwigStack for two twig queries

FIG. 15A

FIG 15B

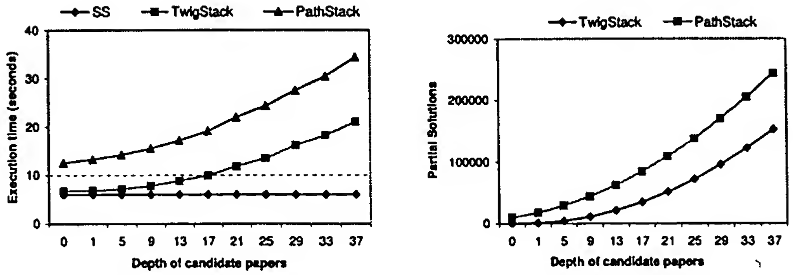
FIG 15C



(a) Execution time (b) Number of solutions (c) Execution time for a complex query

PathStack versus TwigStack for a parent-child twig query

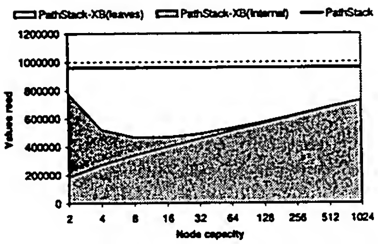
FIG. 16A FIG 16B FIG 16C



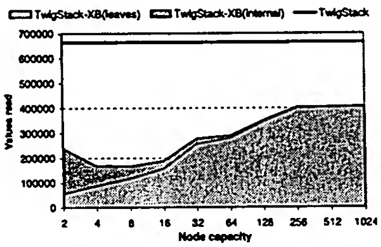
(a) Execution time (b) Number of partial solutions

ᴾathStack versus TwigStack on a real data set

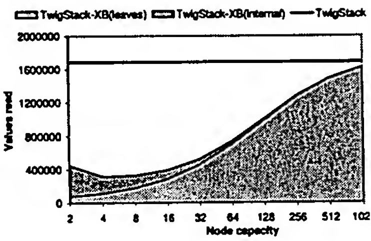
FIG. 17A FIG 17B



(a) Path query



(b) Twig query



(c) Twig query

**Using XB-trees: (a), (b) with synthetic data sets, and (c) with unfolded DBLP data**

FIG. 18A FIG 18B FIG. 18C